ECE 250 Project 3 Hashtables
Design Document

Anthony Xu, UW User ID:a68xu

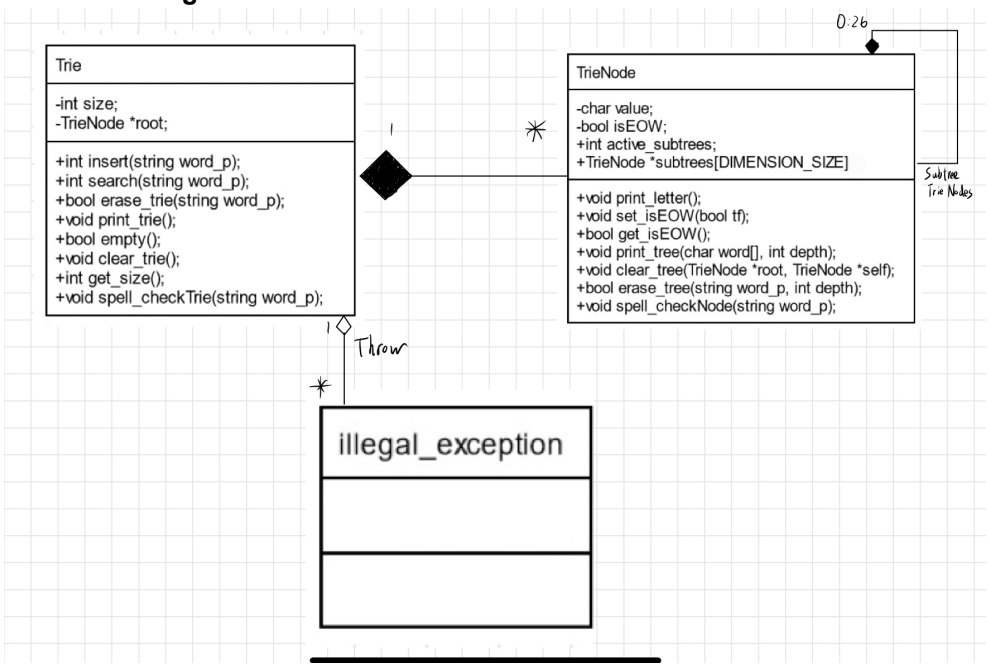
Nov 21 2022

Overview of classes:

Classes used:

- TrieNode
  - Contains a local array of 26 pointers to other TrieNodes.
  - Contains value to indicate if leafnode.
  - contains numerous helper functions for recursively performing operations.
- Trie
  - The Trie contains a root TrieNode and supports all operations required by the project doc.
- illegal_exception
  - Allows for throwing an exception.

UML Class diagram



From: http://www.cs.utsa.edu/~cs3443/uml/uml.html

Details on design decisions

TrieNode Class

```
class TrieNode
{
private:
    char value;
    bool isEOW; // indicate if is leaf

public:
    int active_subtrees;
    TrieNode *subtrees[DIMENSION_SIZE]{}; // 26 ary array storing pointers to Trie
    TrieNode(char value_p);
    ~TrieNode();
    void print_letter();
    void set_isEOW(bool tf);
    bool get_isEOW();
    void print_tree(char word[], int depth);
    void clear_tree(TrieNode *root, TrieNode *self);
    bool erase_tree(string word_p, int depth);
    void spell_checkNode(string word_p);
};
```

Member Variables

- char value variable is an extra measure, it is not used for this project. For a larger project, it can help check if the node is pointed by a pointer in the correct index of the parent node.
- bool isEOW indicates if the variable is a leaf node or not, indicating end of word.
- Active_subtrees indicate how many childs the node has

Member Functions

- constructor initializes all member variables
- Print_letter is for testing purposes, prints the character that the node is supposed to have
- set_isEOW and get_isEOW are getters and setters for isEOW, which indicates the end of a word
- Print_tree can be called on any node, which prints all words that the node as a root and all its subtrees contains in alphabetical order using depth first traversal. The function itself is recursive, passing in the current word that might be incomplete each time and the depth of the current node that was called upon
- Clear_tree allows for the deletion of all the subtrees of a node,recursive
- Erase_tree allows for deleting a word from a node and its subtrees, recursive
- Spellcheck_node allows for spell checking a word from a node and its subtrees,iterative, it calls the print tree function with a already half complete word if there are possible related words.

Trie Class

```
class Trie
{
private:
    int size;
    TrieNode *root;
public:
    Trie();
    ~Trie();
    int insert(string word_p);
    int search(string word_p);
    bool erase_trie(string word_p);
    void print_trie();
    bool empty();
    void clear_trie();
    int get_size();
    void spell_checkTrie(string word_p);
};
```

Member variables:

- Root node and size of the tree

Member Functions:

- Insert traverses the tree and see if the word is already in the tree, if not, insert all the required letters and set the last letter node to end of word
- Search traverses the tree by finding the correct indices of the array according to the input word_p string. If the last letter is found in the tree in the correct position, check to see if it is end of the word.
- Erase_trie calls the erase_tree function on the root node,which then recursively traverses to the end and delete nodes from the leaf node using depth first traversal.
  (if the node corresponding to the last letter of the word is a leaf node and is end of word, delete the node,and return the recursion Otherwise return back to the parent
)
- print_trie calls the print_tree function on the root node, which traverses the whole tree in a recursive manner and print a word if the leaf node is reached
- clear_trie calls the clear_tree function on the leaf node and recursively traverse to the leaf node using depth first traversal and deleting the leaf nodes
- Get_size returns the size of the tree, the size is kept track of each time we insert or delete a word
- spellcheckTrie traverses the node iteratively until it reaches the first letter that does not match the input word, it then prints all related words by calling print_tree on the node. Or, it prints out the correct word, if it is found that the word is in the list during the process.(if the last letter of the word corresponds to a node that is a end of word node)
- Empty checks whether the root node points to no children. It is also ok to use the activeSubtrees variable of the root node to keep track of this, this function is just for testing and debugging purposes, its not used by an member functions

illegal_exception Class

```
class illegal_exception
{
public:
    illegal_exception();
};
```

To allow for throwing an exception of this type

Test Cases

1. When inserting words, check if inserting the same word afterwards results in a failure, check this again after inserting a few different words in between.
2. When inserting multiple words, print the Trie along, and also search for substrings and parent strings of the word we are inserting in. For example after inserting the word “samurai”, search for “samura”, “samuraii”,“samu”,“samuraa”, etc.
3. Test if the total word inputted into the program matches the word count.
4. Testing for erasing the words when the Trie is empty
5. Testing for erasing words that are not in the Trie
6. Testing for erasing words that is a sub word of another word in the trie
7. Testing for erasing words that is a parent string of another smaller sub word in the Trie

8. Testing for checking if the corpus is loaded in, and check if it prints in alphabetical order
9. Testing for illegal arguments
10. Testing for if the first command is not the load command, test all commands, and then load

Run Time

n=M=length of word

- Load
  - O(N)
- Insert
  - O(M)for average case and worst case. The run time is the length of the string that we are inserting, since we need to iterate M levels in all cases. Allocating a new load and setting the last letter to EOW, finding the address to the next node, checking if a node needs a new node for the next letter all takes constant time
- Search
  - O(M) average and worst case.We need to iterate M levels to determine if the word is in the Trie. Finding the address to the next node, checking if a node needs a new node for the next letter all takes constant time.
- erase
  - O(M) average and worst case.We need to iterate M levels to determine if the word is in the Trie, and to delete them recursively. Deletion of a node and setting to nullptr all takes constant time
- Print
  - O(N) we need to visit all nodes in the Trie recursively once. Adding a letter to a string and printing it when reaching end of word takes constant time
- Spellcheck
  - O(N) we need to visit N/26 nodes on average, which is of order N. In the worst case, we need to visit all Nodes if the nodes all start with the same letter, and we call with just that letter, this is also O(N). In best case, the node is no in the list or the first node/letter is the word that we want to spellcheck, which takes constant time
- empty
  - O(1) since I keep track of the Trie size when I perform all operations on the Trie, we can achieve constant run time since the Trie object stores the size of the Trie.
- clear
  - O(N) since we need to visit all nodes in the tree and delete them recursively
- Exit
  - O(N) since we need to visit all nodes in the tree and delete them recursively, and then exit the program